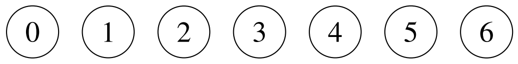
HW 9

1. (20 pts.) **Kruskal's Algorithm.** The order of edges that are added to the MST is as below with the current set(s):

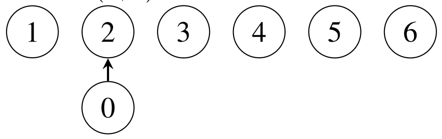
1. (d,f); current sets: $\{a\},\{b\},\{c\},\{d,f\},\{e\}$
2. (a,b); current sets: $\{a,b\},\{c\},\{d,f\},\{e\}$
3. (b,e); current sets: $\{a,b,e\},\{c\},\{d,f\}$
4. (d,e); current sets: $\{a,b,e,d,f\},\{c\}$
5. (e,c); current set: $\{a,b,e,d,f,c\}$

2. (20 pts.) **Disjoint Set Operations.**

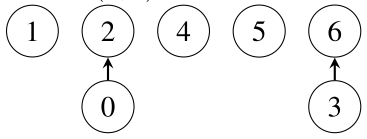
(a) Initial:



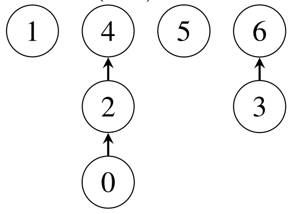
`union`$(0,2)$:



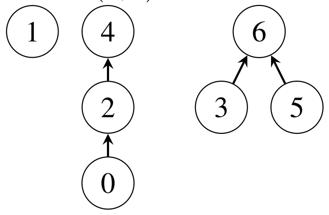
`union`$(3,6)$:



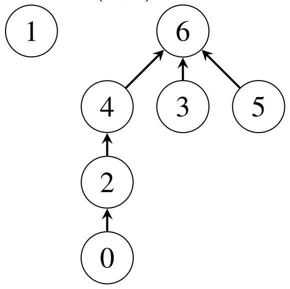
`union`$(2,4)$:



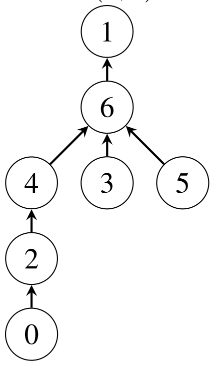
`union`$(5,3)$:



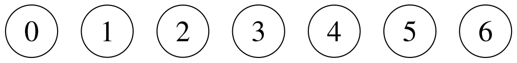
`union`(0,6):



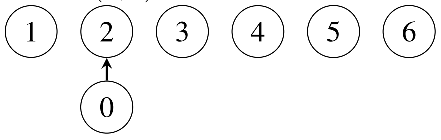
`union`(5,1):



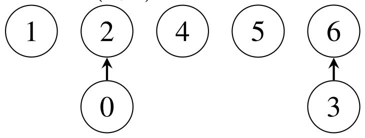
(b) Initial:



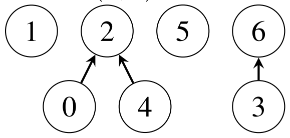
`union`(0,2):



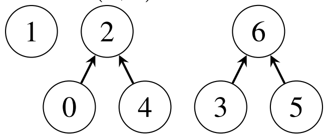
`union`(3,6):



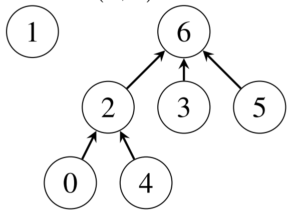
`union`(2,4):



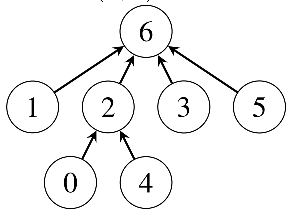
`union`(5,3):



`union`$(0,6)$:



`union`$(5,1)$:



(c) `root` must visit each node between the desired node (here 0) and the root, inclusive. Looking at part (a)'s final tree, this requires visiting 5 of the 7 nodes, (0, 2, 4, 6, and 1). Part (b)'s final tree requires only 3 of the 7 nodes to be visited. (0, 2, and 6).

(d) The worst-case series of unions for part (a) turns the final tree into a single branch, for example, `union`$(0,1)$, `union`$(1,2)$, `union`$(2,3)$, etc. Because every node is on a single path from the root, running `root` on the bottom-most node requires visiting all 7 nodes. However, the worst-case series of unions for part (b) is less clear. Because any merged tree is placed as a direct child of the root in the tree it's merging into, the height can only be increased when merging two trees of equal heights. So, creating a height 3 tree requires at least merging two height 2 trees, or 4 nodes total. Therefore, we do not have enough nodes to construct a height 4 tree, which would require at least 2 height 3 trees, or 8 nodes total. So the worst possible number of nodes on the path from a leaf to the root is 3 for part (b).

3. (20 pts.) **Fractional Knapsack.**

(a) First, we sort the items in descending order of the ratio $\frac{v_i}{w_i}$. Then, we pick items from the beginning of the sorted list until we're either out of items (in which case we're done) or our knapsack does not have enough remaining weight to add the next item. In the latter case, we take as much of the next item as we can until our total weight is exactly W. The running time of this algorithm is dominated by the time it takes to sort the items, which is $O(n \log n)$. The correctness of this algorithm is discussed in part (b).

(b) This greedy algorithm relies on the assumption that picking the item with the best value-to-weight ratio will always be an optimal choice. In the 0-1 Knapsack problem, it is easy to create an example where this assumption fails. For example, $W = 10$, $(v_1, w_1) = (10,5)$, and $(v_2, w_2) = (11,10)$. Item 2 has a worse ratio, but picking item 1 uses only half the available weight, which allows item 2 to achieve a higher total value. Generalizing this idea, we can say that greedily selecting items based only on their v_i and w_i does not always arrive at the optimal solution, as the amount of remaining weight in the knapsack and the other possible selections must be taken into account at each step. In contrast, the Fractional Knapsack allows us to always maximize the weight we select to be equal to W. This removes the possibility that a greedy choice might block better future choices. Any optimal selection must use the entire weight W since all the values are positive. Therefore, the greedy solution that picks the highest value-to-weight ratio, multiplied by W, results in the highest possible value for a weight W knapsack.

4. (20 pts.) **Largest k-Degree Subgraph.** If a node has degree less than k, then neither it nor its edges can be included in G'. Thus, if v has degree less than k, then $G' \subseteq G - v$. This implies the correctness of the following "greedy" algorithm: while G has a node v of degree less than k, set $G := G - v$ and repeat.

To implement this, we first calculate the degree of every node in the graph by finding the number of nodes in each node's adjacency list. Then, we construct a list of nodes with degree less than k by iterating over every node and checking the precomputed degree value. While the list of nodes with degree less than k is not empty, take an element from the list, remove it from the graph, subtract one from the degree of all of its neighbors, and if any of them now have degree less than k, add them to the list. When the list is empty, G' is comprised of the nodes remaining in the graph and the edges between them in E.

This algorithm takes $O(|V| + |E|)$ steps, since each vertex is examined at most once, and there may be a number of subtractions equal to the number of edges.

5. (20 pts.) **MST Edge Reweighting.**

(a) Since the change only increases the cost of some other spanning trees (those including e) and the cost of T is unchanged, it is still an MST.

(b) We include e in the tree, thus creating a cycle. We then remove the heaviest edge e' in the cycle, which can be found in linear time, to get a new tree T'. To prove this, we argue that T' contains a least-weight edge across every cut of G and is hence an MST. Note that since the only changed edge is e, $T \cup \{e\}$ already includes a least-weight edge across every cut. We only removed e' from this. However, any cut crossed by e', must also be crossed by at least one more edge of the cycle, which must have weight less than or equal to e'. Since this edge is still present in T', it contains a least-weight edge across every cut. Thus, T' is an MST.

(c) The tree is still an MST if the weight of an edge in the tree is reduced. Hence, no changes are required.

(d) We remove e from the tree to obtain two components, and hence a cut. We then include the lightest edge across the cut to get a new tree T'. We can now "build up" T' using the cut property to show that it is an MST. Let $X \subseteq T'$ be a set of edges that is part of some MST, and let $e_1 \in T' \setminus X$. Then, $T' \setminus \{e_1\}$ gives a cut which is not crossed by any edge of X and across which e_1 is the lightest edge. Hence, $X \cup \{e_1\}$ is also a part of some MST. Continuing this, we can grow X to $X = T'$, which must be then an MST.

Rubric:

Problem 1, 20 pts

- 2 pts for each correct edge stated in the correct order
- 2 pts per step for the vertex sets (order of elements in each set does not matter)

Problem 2, 20 pts

(a) 6pts, 1 for each union forest.
(b) 6pts, 1 for each union forest.
(c) 4pts for the correct answer, (a) requires 5, (b) requires 3.
(d) 4pts for the correct answer, (a) requires at most 7, (b) requires at most 3.

Problem 3, 20 pts

(a) 12 pts for the correct ratio-based algorithm.
(b) 8 pts for explaining 0-1 can waste space, but Fractional always fills the space, so best ratio is optimal.

Problem 4, 20 pts

- 12 pts for designing correct algorithm
- 4 pts for proof of correctness (just one or two sentences would be enough for this problem)
- 4 pts for showing why the algorithm runs in $O(|V|+|E|)$. Note that the algorithm should be correct in order to get the point of this part.

Problem 5, 20 pts

(a) this part is worth 4 pts, only saying that ”it is still an MST” is enough.
(b) this part is worth 6 pts, designing correct algorithm (or equivalently updating MST tree correctly) is worth 4 pts, and proof of correctness is worth 2 pts.
(c) this part is worth 4 pts, only saying that ”no update is needed” would be enough.
(d) this part is worth 6 pts, designing correct algorithm (or equivalently updating MST tree correctly) is worth 4 pts, and proof of correctness is worth 2 pts.